Diferentes Tipos de Datos

D3 es muy flexible en cuanto a datos de entrada. Este tema introduce las estructuras de datos que se usan comúnmente en JavaScript y D3.

Variables

Una variable es un dato, el bloque más pequeño de información. La variable es el cimiento para todas las demás estructuras de datos, que simplemente son configuraciones diferentes de las variable.

Si usted está hasta ahora empezando con JavaScript, sepa que es un lenguaje *débilmente tipado*, lo que quiere decir que no es necesario especificar de antemano el *tipo* de información que se va a almacenar en una variable. Muchos otros lenguajes, como Java (que es totalmente diferente de JavaScript!), requieren que se declare el tipo de variable, tal como `int, float, boolean, o String`

```
//Declaración de variables en Java
int number = 5;
float value = 12.3467;
boolean active = true;
String text = "Crystal clear";
```

JavaScript, en cambio, le asigna un *tipo* automáticamente a la variable con base en el tipo de información que se le asigne. (Tome nota de que ’ ’ y ” ” son indicadores de valores de una cadena de caracteres. Yo prefiero las comillas dobles ” “, pero algunas personas prefieren las comillas sencillas ”.

```
//Declaración de variables en JavaScript
var number = 5;
var value = 12.3467;
var active = true;
var text = "Crystal clear";
```

Qué aburrido - `var, var, var, var`- sin embargo es útil a medida que declaramos y nombramos variables antes de que sepamos qué tipo de datos serán almacenados allí. Uno puede también cambiar el tipo de dato sobre la marcha sin que JavaScript se enloquezca.

```
var value = 100;
value = 99.9999;
value = false;
value = "Es imposible que esto funcione";
value = "Ahh, si funciona!  No saca errorrrrr!";
```

Arreglos

Un arreglo es una secuencia de valores, que se almacena muy convenientemente dentro de una sola variable.

Manejar datos relacionados en diferentes variables es muy ineficiente:

```
var numberA = 5;
var numberB = 10;
var numberC = 15;
var numberD = 20;
var numberE = 25;
```

Al reescribir esto como un arreglo, se vuelven mucho más sencillos estos valores. Los corchetes definen un arreglo y las comas separan cada uno de los valores:

```
var numbers = [ 5, 10, 15, 20, 25 ];
```

Los arreglos son un componente intrínseco de las visualizaciones de datos, por consiguiente es necesario entenderlos bien. Se puede acceder al valor almacenado en un arreglo, utilizando notación de corchetes:

```
numbers[2] //Devuelve 15
```

El numeral dentro del corchete se refiere a la posición correspondiente en el arreglo. Recuerde que las posiciones del arreglo comienzan en cero, entonces la primera posición es 0, la segunda es 1 y así sucesivamente.

```
numbers[0]  //Devuelve 5
numbers[1]  //Devuelve 10
numbers[2]  //Devuelve 15
numbers[3]  //Devuelve 20
numbers[4]  //Devuelve 25
```

Algunas personas encuentran útil pensar en arreglos en términos espaciales, como si tuvieran filas y columnas, como una hoja de cálculo:

```
ID | Valor
```

```
0 | 5
1 | 10
2 | 15
3 | 20
4 | 25
```

Los arreglos pueden contener cualquier tipo de datos, no solamente enteros.

```
var percentages = [ 0.55, 0.32, 0.91 ];
var names = [ "Ernie", "Bert", "Oscar" ];

percentages[1]  //Returns 0.32
names[1]        //Returns "Bert"
```

De qué están hechos los Arreglos. Función for()

La visualización de datos basada en código no sería posible sin arreglos y el poderoso loop `for()`. Juntos, conforman el dúo dinámico de los “geeks” de datos. (En caso de que

usted no se considere un “data geek, le recuerdo que está leyendo el documento titulado “Tipos de datos.”).

Un arreglo organiza muchos datos en un lugar conveniente. Luego, la función `for()` puede rápidamente recorrer todos los datos del arreglo y ejecutar alguna acción sobre ellos – tal como mostrar su valor de manera visual. D3 por lo general maneja el recorrido (looping) con su método mágico `data()`, pero es importante poder escribir nuestros propios loops.

No voy a describir la mecánica de los loops `for()` acá; esto da para otro tutorial entero por separado. Pero tome nota de este ejemplo, el cual recorre los valores del arreglo `numbers` de arriba.

```
for (var i = 0; i < numbers.length; i++) {
console.log(numbers[i]);  //Imprime el valor a la consola
}
```

Ve ese `numbers.length`? Esa es la parte bella. Si `numbers` tiene diez posiciones, el loop correrá diez veces. Si tiene diez millones de posiciones… sí, usted ya sabe. Para esto es que son buenas las computadores: para recibir un grupo de instrucciones y ejecutarlas una y otra vez. Y esto se encuentra en el corazón de por qué son tan satisfactorias las visualizaciones de datos – uno diseña y codifica el sistema de visualización y el sistema responde acorde, aún cuando se le entrega diferentes datos. Las reglas de mapeo del

sistema son consistentes, aún cuando los datos no lo son.

Objetos

Los arreglos son excelentes para listas simples de valores, pero cuando se tienen conjuntos complejos de datos, es deseable almacenar los datos en un objeto. Para nuestro propósito, se debe pensar en un objeto de JavaScript como si fuera una estructura de datos hecha a la medida. Se usan las entre llaves {} para identificar a un objeto. Dentro de las llaves, se incluyen índices y valores. Dos puntos `:` separan cada índice de su valor, y una coma separa cada par de índice/valor.

```
var fruta = {
    tipo: "uva",
    color: "rojo",
    cantidad: 12,
    dulce: true
};
```

Para referenciar cada valor, se utiliza la *notación de puntos*, especificando el nombre del índice.

```
fruta.tipo      //Devuelve "uva"
fruta.color     //Devuelve "rojo"
```

```
fruta.cantidad  //Devuelve 12
fruta.dulce     //Devuelve true
```

Piense en el valor como “perteneciendo” al objeto. Mire, una fruta. “Qué tipo de fruta es? puede que se pregunte. Y de veras, `fruta.tipo` es `"uva"`. Y es dulce? Sí, definitivamente, puesto que `fruta.dulce` es `true`.

Objetos y Arreglos

Es posible combinar estas dos estructura y crear arreglos de objetos, u objetos de arreglos, u objetos de objetos, o básicamente cualquier estructura que tenga sentido para su conjunto de datos.

Digamos que hemos adquirido otras frutas y que queremos expandir nuestro catálogo. Utilizamos los corchetes `[]` en la parte de afuera par indicar que es un arreglo, seguido por las entre llaves `{}` y notación de objetos en la parte de adentro, con una coma separando a cada objeto.

```
var frutas = [
{
    tipo: "uva",
    color: "rojo",
    cantidad: 12,
    dulce: true
```

```
  },
  {
      tipo: "kiwi",
      color: "cafe",
      cantidad: 98,
      dulce: true
  },
  {
      tipo: "banano",
      color: "amarillo",
      cantidad: 0,
      dulce: true
  }
  ];
```

Para obtener acceso a estos datos, debemos seguir el sendero de los índices hacia los valores que buscamos. Recuerde, [] significa arreglo, y {} significa objeto.`frutas` es un arreglo, entonces utilizamos la notación de corchetes primero para especificar el índice del arreglo.

`fruta[1]`

Seguido, cada elemento del arreglo es un objeto, entonces se añade un punto y un índice:

`fruta[1].cantidad //Devuelve 98`

Acá está el mapa de cómo acceder a todos los valores del arreglo de objetos denominado `frutas`:

```
frutas[0].tipo     == "uva"
frutas[0].color    == "rojo"
frutas[0].cantidad == 12
frutas[0].dulce    == true

frutas[1].tipo     == "kiwi"
frutas[1].color    == "cafe"
frutas[1].cantidad == 98
frutas[1].dulce    == true

frutas[2].tipo     == "banano"
frutas[2].color    == "amarillo"
frutas[2].cantidad == 0
frutas[2].dulce    == true
```

Ciertamente, sabemos que `frutas[2].cantidad` bananos.

JSON

En algún punto de su carrera de D3, encontrará la Notación de Objetos de Javascript. Puede leer los detalles, sin embargo JSON es básicamente un tipo de sintaxis específico para organizar datos como objetos de JavaScript. Esta sintaxis se ha optimizado para uso con JavaScript (obviamente) y llamadas AJAX, por eso es que verá muchos de los APIs de la red botando datos en formato JSON. Es más rápido y fácil de descomponer con JavaScript que XML y por supuesto, D3 funciona bien con este formato.

Todo esto, y en resumen no se ve mucho más raro que lo que ya hemos visto:

```
var jsonFruta = {
    "tipo": "uva",
    "color": "rojo",
    "cantidad": 12,
    "dulce": true
};
```

La única diferencia acá es que nuestros índices ahora están rodeados por comillas ” “, de tal forma que todos los valores son cadenas de caracteres.

GeoJSON

Así como JSON es solamente un formalismo para la sintaxis de objetos de JavaScript existente, GeoJSON es la sintaxis formal de objetos JSON, optimizada para almacenar

información geográfica. Todos los objetos GeoJSON son objetos JSON, y todos los objetos JSON son objetos de JavaScript.

GeoJSON puede almacenar puntos en un espacio geográfico (típicamente coordenadas longitud/latitud), pero también formas (como líneas y polígonos) y otro tipo de representaciones espaciales. Si tiene bastante información geográfica, vale la pena descomponerla en formato GeoJSON para optimizar su uso en D3.

Entraremos en detalles acerca de GeoJSON cuando hablemos de mapas geográficos, pero por el momento, es suficiente con saber que un dato simple en formato GeoJSON puede representarse así:

```
var geodata = {
    "type": "FeatureCollection",
    "features": [
        {
            "type": "Feature",
            "geometry": {
                "type": "Point",
                "coordinates": [ 150.1282427, -24.471803 ]
            },
            "properties": {
                "type": "town"
            }
        }
```

```
    ]
  };
```

(Si es confuso, la longitud siempre se pone antes que la latitud. Tiene que acostumbrarse a pensar en términos de longitud/latitud en vez de latitud/longitud).

Siguiente: Diagrama de Barras –>

Este tutorial fue traducido por Gabriel Coch
Todo el contenido fue desarrollado por y le pertenece a Scott Murray